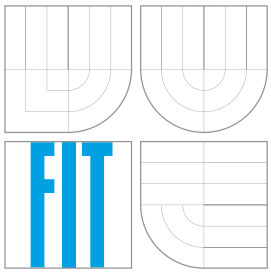
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

POKROČILÉ NÁSTROJE PRO ODPOSLECHY NA SÍŤOVÉ SONDĚ

ADVANCED TOOLS FOR LEGAL INTERCEPTION ON NETWORK PROBE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN VRÁNA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV BARTOŠ

BRNO 2014

Abstrakt

Tato práce se zabývá návrhem a implementací části systému pro zákonné odposlechy síťového provozu. Navrhovaný systém počítá se zpracováním vysokorychlostního provozu s propustností až 100 Gb/s. Výsledný systém bude využívat hardwarové akcelerace karty s podporou konceptu Software Defined Monitoring (SDM). Samotný software je však navržen tak, aby umožňoval zpracování co nejvyššího množství rámců i bez hardwarové podpory.

Abstract

This thesis describes design and implementation of one the parts of lawful interception system for intercepting network traffic. Designed system will be used for processing traffic with at maximum throughput of 100 Gbps. Resulting system will use hardware acceleration with Software Defined Monitoring (SDM) features. Software itself is designed to be able to process as many netwrok frames as possible even without hardware acceleration.

Klíčová slova

zákonné odposlechy, Software Defined Monitoring, zpracování vysokorychlostního síťového provozu, hardwarová akcelerace

Keywords

lawful interception, Software Defined Monitoring, high-speed network traffic processing, hardware acceleration

Citace

Roman Vrána: Pokročilé nástroje pro odposlechy na síťové sondě, bakalářská práce, Brno, FIT VUT v Brně, 2014

Pokročilé nástroje pro odposlechy na síťové sondě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Václava Bartoše.

.....
Roman Vrána
20. května 2014

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Václavu Bartošovi a Ing. Martinu Žádníkovi, Ph. D., za pomoc poskytnutou při tvorbě této práce. Dále bych také rád poděkoval výzkumné skupině ANT a Sec6Net za možnost podílet se na vývoji jejich systému pro zákonné odposlechy.

Obsah

Kapitola 1

Úvod

Internet patří v současné době k nejrozšířenějšímu médiu ke komunikaci mezi lidmi (textové nebo hlasové) nebo poskytování služeb. Na druhou stranu se však na Internet přesouvá i kriminální činnost. Může se jednat o sdílení nelegálního či trestně postihnutelného obsahu, využívání komunikačních služeb k organizování trestné činnosti nebo vlastní kybernetické útoky. S rozšířením sociálních sítí k tomu lze přidat zneužívání těchto sítí k podvodům nebo kyberšikaně, vydírání nebo vyhrožování. Kvůli množství přenášených dat jsou tyto činnosti jsou v síti Internetu obtížně dohledatelné a navíc je obtížné spojit tato data s případným pachatelem, jelikož není možné k pachateli jednoznačně přiřadit danou Internetovou identitu. K prokazování trestné činnosti však existují standardy pro vývoj systému pro zákonné odposlechy na síti (Lawful Interception System, dále jen LI systém). Ty specifikují, jak komunikaci sledovat a následně identifikovat její zdroj. Využití Internetu však nadále vzrůstá a s ním se i zvyšují požadavky na rychlost a kvalitu přenosu. V současné době jsou již dostupná hardwarová zařízení schopná pracovat na rychlostech v řádu desítek až stovek Gb/s. Pro účely odposlechu tak potřebujeme řešení, které je schopné zpracovat provoz na takto vysoké rychlosti přenosu bez ztráty jediného rámce. Neúplný odposlech by jinak mohl být soudním orgánem zamítnut. Jedním z řešení je možné využít kombinace software, který se bude starat o samotné zpracování odposlouchávané komunikace (přiřazení k odposlouchávanému datovému toku, propojení dat s identitou atd.), a hardware, který bude poskytovat základní předzpracování síťového provozu např. propuštění pouze zájmového provozu do software. Řešení na čistě softwarové bázi je pro vysoké rychlosti nedostatečné.

Tato práce je zaměřena na vývoj části tohoto systému v podobě softwarového filtračního modulu, který bude sloužit pro odchycení zájmového provozu a jeho následné předání k dalšímu zpracování. Tento modul bude dále využívat hardwarové akcelerace k tomu, aby mohl nezájmový provoz zahodit ještě před vlastním zpracováním. V práci bude popsána implementace vlastního zpracování přijímaného provozu a postup jeho filtrace a datové struktury využité pro ukládání pravidel nutných pro klasifikaci přijatých dat. Dále je v práci zahrnuto i testování implementovaného řešení v laboratorních podmínkách s generovaným síťovým provozem.

Ve vlastní práci bude nejprve popsáno zapouzdření síťového provozu a referenční síťový model a souvislost se zpracováním odposlouchávaných dat. Dále budou uvedeny prostředky dostupné pro realizaci tohoto systému včetně již existujících implementací podobných systémů. Ve třetí kapitole bude popsána samotná implementace filtrační jednotky a modulů

pro její správu. Čtvrtá kapitola se bude zabývat testováním implementovaného řešení v laboratorních podmínkách na schopnosti zpracování provozu. V závěru pak bude provedeno zhodnocení dosažených výsledků a případné návrhy na další vylepšení implementace nebo další využití.

Kapitola 2

Teoretický rozbor a dostupné prostředky

2.1 Síťové modely a jejich vazba na odposlechy

Před vlastní problematikou síťových odposlechů je vhodné nejprve vysvětlit modely, na kterých je postaveno fungování síťové komunikace. Na ně bude v práci dále odkazováno při popisu vývoje jednotlivých částí a testování výsledného celku. Jedná se především o referenční model ISO/OSI a model TCP/IP využívaný v současném Internetu. Podrobnější popis těchto modelů lze najít například v [13].

2.1.1 Referenční model ISO/OSI

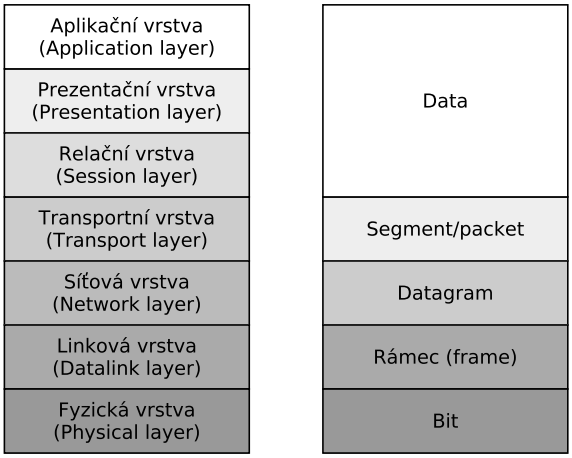
Referenční model OSI (Open Systems Interconnect Reference Model) vydaný v 1987 Mezinárodní standardizační organizací ISO slouží jako soubor standardů pro komunikaci mezi zařízeními po síťovém médiu. Model slouží jako hlavní vzor pro implementaci mnoha současných síťových protokolů. Implementace modelu je v praxi provedena pouze částečně. Plná implementace zatím nikdy nebyla uvedena do provozu.

Model ISO/OSI je složen ze sedmi vrstev, kde každá definuje protokoly a služby příslušné funkčnosti dané vrstvy. Funkce jsou definovány pro přenos dat mezi procesy na stejné vrstvě modelu. Model však nedefinuje jeden protokol ale pouze způsoby komunikace, které musí protokoly na příslušné vrstvě modelu implementovat. Každá vyšší vrstva modelu může využívat služeb nižší vrstvy, aniž by musel znát přesnou implementaci. Mimo jiné model definuje i základní datové jednotky (PDU - Process Data Unit) pro jednotlivé vrstvy. 2.1

Vrstvy modelu ISO/OSI

Fyzická vrstva definuje standardy a fyzické vlastnosti přenosového média. Těmi jsou např. napěťové úrovně pro logickou 0 a 1, rozmístění pinů konektoru, přenosové charakteristiky apod. Příkladem protokolů pro fyzickou vrstvu jsou IEEE 802.3 (CSMA/CD) a 802.11 (Wi-Fi), FDDI, Token Ring (802.5), RS-232C a další.

Linková vrstva popisuje přenos dat po datovém spojení, vytváření datových rámců a jejich adresaci. Standardy vztahující se k této vrstvě jsou Ethernet 802.3 a 802.2, Frame Relay, PPP aj.



Obrázek 2.1: Model ISO/OSI a odpovídající datové jednotky

Síťová vrstva slouží k adresování a směrovaní dat po síti. Dále jsou zde definice pro spojované služby a protokoly a nespojované služby a protokoly. Na síťové vrstvě pracují také směrovací protokoly. Z v praxi použitých implementacích této vrstvy lze uvést i protokoly IP z modelu TCP/IP nebo IPX z modelu Novell NetWare.

Transportní vrstva se stará o spolehlivý přenos dat mezi komunikujícími uzly. Podle standardu ISO jsou zde definovány třídy pro klasifikaci služeb podle spolehlivosti přenosu. Na této vrstvě jsou dále implementovány spojované a nespojované transportní protokoly.

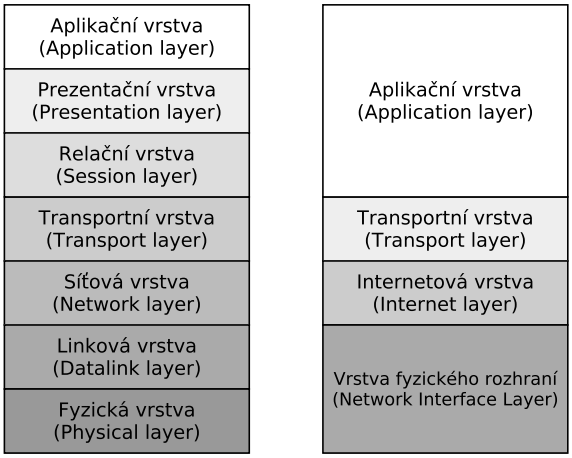
Relační vrstva zajišťuje udržování relace mezi komunikujícími body. Služby této vrstvy se starají založení a zrušení relace, obsluhu dialogů při poloduplexní komunikaci, synchronizaci apod.

Prezentační vrstva poskytuje standardy pro prezentaci dat mezi aplikacemi a architekturami. Zahrnuje datové formáty jako ASCII, EBCDIC, binární reprezentace dat, kompresi dat nebo kódování. Služby a protokoly jsou definovány ve standardech ISO 8822 a 8823. Dále je na této vrstvě definována abstraktní syntaktická notace ASN.1 (standard ISO 8824) pro reprezentaci dat např. v síťových hardwarových zařízeních.

Aplikační vrstva definuje komunikující aplikace a uživatelské procesy. Jsou zde zahrnuty např. služby pro obecné aplikace CASE (standardy ISO 8649 a 8650) jako elektronická pošta MHS (standard ITU-T X.400), adresářové služby (standardy ISO 9594, ITU-T X.500), virtuální terminál (standardy ISO 9040 a 9041) nebo přenos, přistup a zpracování souborů (FTAM - File Transfer Access and Management).

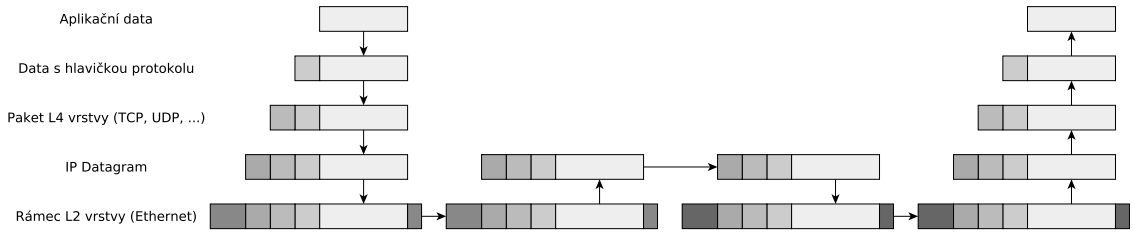
2.1.2 Model TCP/IP

Model TCP/IP byl navržen v 60. letech 20. století se vznikem sítě ARPAnet (základ současného Internetu). Tato síť byla oficiálně uvedena do provozu v roce 1975, kdy již byly vyvinuty základní protokoly tohoto modelu. Protokoly TCP a IP pak byly v roce 1981 standardizovány. V roce 1983 byly tyto protokoly implementovány v unixovém operačním systému Berkeley Unix (BSD).



Obrázek 2.2: Modely ISO/OSI (vlevo) a TCP/IP (vpravo)

Architektura modelu TCP/IP je proti ISO/OSI výrazně jednodušší. Tento model spojuje některé vrstvy modelu ISO/OSI do jedné. Jedná se spojení aplikační, prezentační a relační vrstvy ISO/OSI modelu do aplikační vrstvy modelu TCP/IP a fyzické a linkové vrstvy do vrstvy fyzického rozhraní, která je součástí implementace síťového hardware. Srovnání lze vidět na obrázku 2.2. Implementace modelu je pak rozdělena do tří částí. Vrstva fyzického rozhraní, jak již bylo zmíněno výše, je implementována v síťovém rozhraní hardwarové karty (NIC) a jejím ovladači. Internetová (síťová) a transportní vrstva jsou pak implementovány v síťovém modulu jádra operačního systému. Nejvyšší vrstvu pak implementují uživatelské aplikace. Příklad komunikace je pak znázorněn na obrázku 2.3 s odpovídajícím zapouzdřením dat do správných PDU. K přenášeným datům jsou postupně připojovány hlavičky

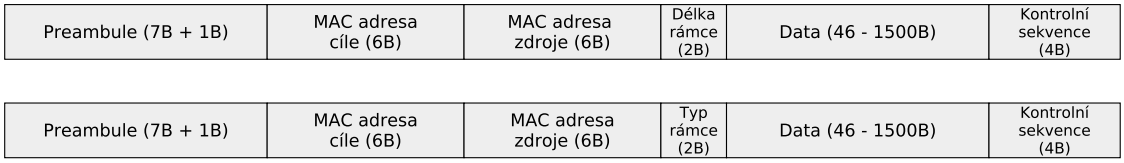


Obrázek 2.3: Znázornění komunikace v modelu TCP/IP

jednotlivých protokolů (aplikační hlavička, hlavička protokolu L4 vrstvy, hlavička protokolu L3 vrstvy) a před odesláním po přenosovém médiu jsou zapouzdřena do rámce L2 vrstvy (nejčastěji Ethernet) a odeslána k cíli. Uprostřed obrázku je znázorněno zpracování rámce směrovačem, kdy je pro přístup k hlavičce protokolu L3 vrstvy využívané ke směrovaní odstraněn rámec a před odesláním je s úpravami vrácen zpět (úpravou je většinou zamýšlena změna zdrojové adresy za adresu výstupního rozhraní). V cíli je pak rámec opačným procesem zpracován až na úroveň aplikačních dat.

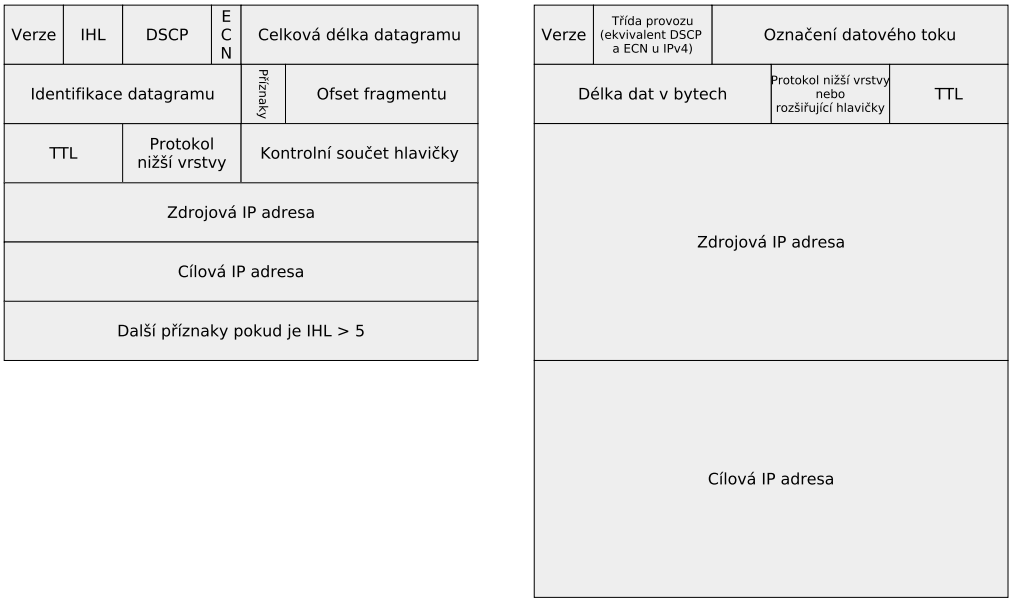
Vrstvy modelu TCP/IP

Vrstva fyzického rozhraní slučuje funkci fyzické a linkové vrstvy modelu ISO/OSI. Definuje tedy standardy pro fyzická přenosová média a zajišťuje funkce pro přístup k nim. Dále se také stará o zapouzdření dat do datových rámců. Příkladem standardů jsou Ethernet 802.2 nebo 802.3, Frame Relay, Token Ring, X.25 aj.



Obrázek 2.4: Ukázka rámce formátu Ethernet 802.3 (nahoře) a Ethernet II (dole)

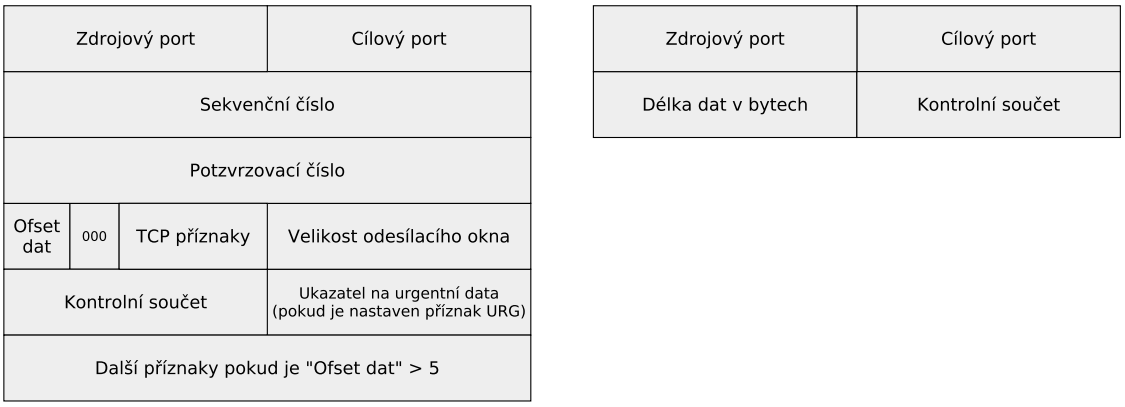
Internetová vrstva zajišťuje vytvoření datagramu, jeho adresování a následně směrování v síti k cílovému bodu způsobem **best-effort delivery** (doručení s největším úsilím). Snaží se tedy najít co nejvhodnější cestu. Jestliže dojde ke ztrátě či poškození datagramu, zdroj je o tomto stavu informován avšak opětovným přenos musí zajistit zdroj samotný. Internetovou vrstvu nejčastěji obsazují protokoly IP, ARP (Adress Resulution Protocol), RARP (Reverse ARP), ICMP (Internet Control Message Protocol pro řízení toku) a IGMP (Internet Group Message Protocol pro správu multicastových spojení). Pro adresování na této vrstvě slouží IP adresy ve verzi 4 o délce 32 bitu nebo ve verzi 6 o délce 128 bitů. Vzor IP hlaviček je znázorněn na obrázku 2.5.



Obrázek 2.5: Hlavičky IPv4 [15] (vlevo) a IPv6 [4] (vpravo)

Transportní vrstva poskytuje logické spojení mezi komunikujícím procesy na zdrojové a cílové stanici za pomocí čísel portů. Posílaná data navíc dělí do menších celků (paketů),

které odesílá po síti. Mimoto ještě zajišťuje mechanismy ustavení spojení nebo řízení toku dat či spolehlivý přenos (protokol TCP). Základní protokoly používané na této vrstvě jsou TCP (Transmission Control Protocol) a UDP (User Datagram Protocol). TCP přebírá aplikační data jako souvislý proud a ve formě paketů jej posílá k cíli. Cíl po obdržení dat provede seřazení paketů podle sekvenčních čísel. TCP je schopen vlastního řízení toku, zahlcení a potvrzování přijetí dat. UDP se naopak snaží doručit data co nejrychleji i za cenu nespolehlivého přenosu. Případně mechanismy pro potvrzování nebo řazení dat si musí uživatelské aplikace zajistit samy. Ve výsledku je UDP spojení rychlejší než TCP, ale při přetížení sítě jsou ztráty dat nevyhnutelné. Hlavičky obou protokolů jsou pak na obrázku 2.6.



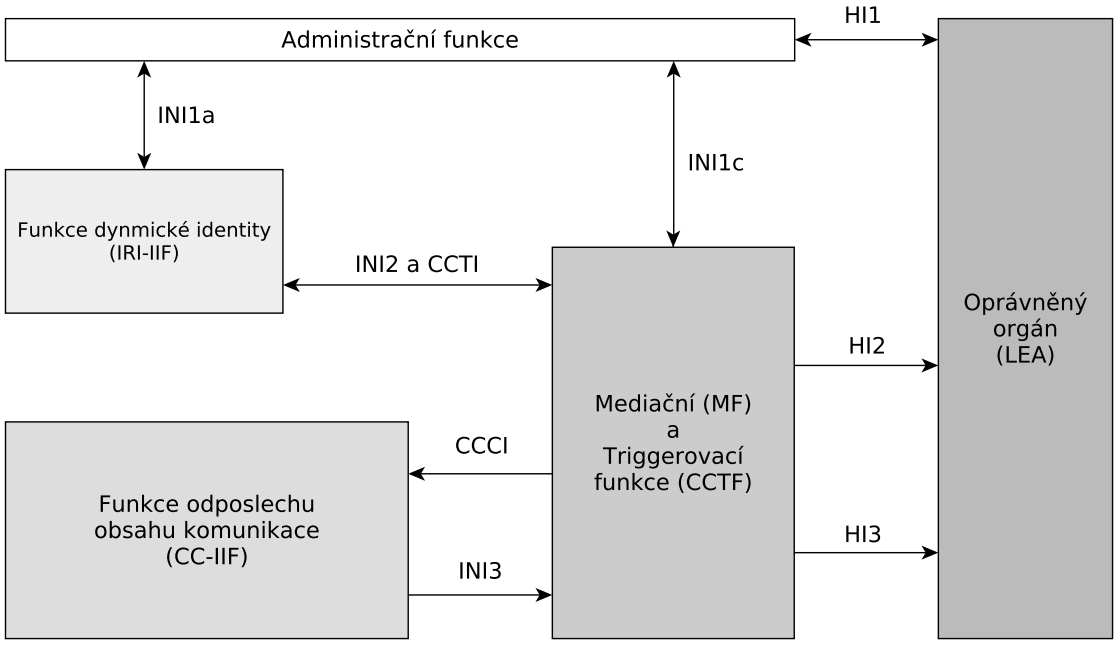
Obrázek 2.6: Hlavičky TCP (vlevo) a UDP (vpravo)

Aplikační vrstva je tvořena samotnými komunikujícími procesy. Tato vrstva poskytuje prostředky pro zpracování dat, kódování, řízení relací a reprezentaci dat. Mezi aplikační protokoly patří například FTP (File transfer Protocol), HTTP (Hypertext Transfer Protocol), Telnet, SSH (Secured Shell), ale i DNS (Domain Name System), DHCP (Dynamic Host Configuration Protocol) nebo směrovací protokoly OSPF (Open Shortest Path First) a RIP (Routing Information Protocol).

Z hlediska síťových odposlechů jsou pro implementovanou část důležité vrstvy L3 a L4 modelu ISO/OSI resp. internetová a transportní vrstva v TCP/IP. Na těchto vrstvách se totiž nacházejí položky používané k identifikaci datového toku a jeho následné klasifikaci k příslušnému toku. Jedná se o obě IP adresy, L4 porty a protokol L4 vrstvy.

2.2 Sec6Net Lawful Interception System (SLIS)

V této sekci je popsán systém pro zákonné odposlechy vyvíjený skupinou Sec6Net [14], pro který je vyvíjen v práci uvedený modul. Na tomto systému bude vysvětlena funkce celého systému pro zákonné odposlechy podle norem ETSI [9, 8, 6, 5, 7]. Vlastní systém je dále inspirován návrhem LI systémů společností Cisco popsaném v RFC 3924 [1]. Blokové schéma systému je uvedené na obrázku 2.7. Ten ukazuje jednotlivé hlavní části LI systému a rozhraní určené pro komunikaci mezi nimi.



Obrázek 2.7: Blokové schéma LI systému

Popis částí LI systému

Administrační funkce AF složí ke správě vlastních odposlechů. V okamžiku obdržení požadavku na odposlech od oprávněného orgánu (LEA) přes rozhraní HI1 funkce nejprve zkontroluje správnost uvedených údajů (správnost časových údajů, přítomnost orgánu v systému, unikátnost identifikátoru LIID atd.). Pokud je vše v pořádku, je požadavek zařazen do fronty čekajících požadavků. AF dále provádí správnou inicializaci a ukončení vlastního odposlechu. AF dále zajišťuje konfiguraci dalších bloků pomocí rozhraní INI1, konkrétně INI1a pro blok IRI-IIF a INI1c pro blok MF a CCTF.

Mediační a triggerovací funkce (MF a CCTF) zabezpečuje zpracování zachycených dat a jejich předání oprávněným orgánům. Tento blok přebírá metadata o odposlouchávané komunikaci od bloku IRI-IIF a vlastní zachycená data od bloku CC-IIF. Uvedená data pak po zpracování předá pomocí rozhraní HI2 respektive HI3.

Funkce dynamické identity (IRI-IIF) sleduje identitu odposlouchávaného cíle a zpracovává její změny v průběhu odposlechu. Jedná se především o změny jako je dynamické přidělení adresy pomocí DHCP nebo identifikace pomocí protokolu 802.1X (RADIUS). Dále tento blok vytváří metadata o odposlouchávaném cíli jako je začátek a konec spojení, přidělení identity, vypršení platnosti identity apod. Tato metadata pak odesílá do MF rozhraním INI2.

Funkce odposlechu obsahu komunikace (CC-IIF) slouží ke sledování síťového provozu a zachytávání dat na základě zaznamenaných požadavků. Konfigurace bloku probíhá přes blok MF a CCTF rozhraním CCCI. Zachycená data jsou následně do tohoto bloku odesílána přes rozhraní INI3.

2.3 Koncept SDM – Software Defined Monitoring

Koncept Software Defined Monitoring (dále SDM) [11] vytváří kombinaci softwarové a firmwarové platformy pro zpracování síťových toků. Vlastní myšlenka vychází z velmi podobného konceptu Software Defined Networking (SDN), který je vyvíjen pro využití v současných síťových architekturách. Základní myšlenkou je předzpracování zachycených dat již v hardwaru na základě pravidel definovaných softwarem. Data mohou být například klasifikována jako nezájmová a hardware je tedy může rovnou zahazovat nebo může software požadovat agregaci některých dat či vzorkování.

Implementace SDM je složena z firmwarové a softwarové části. Firmwarová část přijímá síťový provoz a provádí jeho předzpracování na základě pravidel získaných od softwarové části. Klasifikátor provádí vyhodnocení těchto pravidel a na jejich základě pak rozhoduje, jak má získané rámce náležící pravidlu zpracovat. Může například provádět jejich agregaci do informaci o síťovém toku jako celku, nebo může tato data rovnou zahodit, aby software nebyl zatěžován nezájmovými daty. Do softwaru pak jsou odesílána data v takové podobě, v jaké jsou vyžadována. Firmware může navíc provádět generovaní unifikovaných hlaviček pro software.

Softwarová část se následně stará o vlastní zpracování zachycených dat. Zde mohou být použity aplikace sloužící například k exportu dat nebo k jejich dalšímu zpracování. Softwarová část dále vykonává vyhodnocení pravidel pro klasifikátor firmwarové části. Pokud totiž aplikace obdrží data, o která nemá zájem, je tato skutečnost předána do softwarového řadiče SDM. Ten slouží jako prostředník v komunikaci mezi softwarovou a firmwarovou částí. Data do aplikací jsou pak uchovávána v kruhových seznamech do doby, než jsou aplikací zpracována. Firmware může poskytovat více logických výstupů a každému z nich bude náležet právě jeden tento seznam. Díky tomu může být připojeno ke stejnému firmware více aplikací, nebo jedna aplikace může provádět vícevláknové zpracování dat, kde jednomu vláknu bude náležet jeden logický kanál.

2.4 TRAP

Rozhraní TRAP (TRaffic Analysis Platform) bylo původně vyvinuto v rámci projektu Liberouter[1]jako podvozek pro Framework Nemea [2]. Tento framework slouží k proudové analýze síťového provozu pomocí dat ve formátu IPFIX nebo Cisco Netflow. Knihovna TRAP v něm zajišťuje jednotné komunikační rozhraní mezi jednotlivými moduly vytvořené v tomto frameworku. Umožňuje komunikaci mezi procesy za pomocí schránek jak BSD tak UNIX, nebo komunikaci pomocí sdílené paměti. Díky tomu, že knihovna není pevnou součástí tohoto frameworku, je možné ji využít k implementaci komunikace mezi jednotlivými procesy v bloku CC-IIF. Jediným požadavkem je, aby rychlost přenosu dat byla dostatečná pro předávání zpráv pro nastavení filtrovacích tabulek a následně předzpracovaných rámců. Z testů platformy uvedené v technické zprávě [2] je platforma TRAP schopná přenést až 7 000 000 zpráv o velikosti 66 bytů při použití rozhraní TCP. Při použití UNIX socketů tato hodnota přesahuje 8 000 000 zpráv. Pro velikosti přesahující 1 kilobyte přenosová rychlost klesá těsně pod 1 000 000 zpráv pro TCP a dosahuje cca 1 500 000 pro UNIX sockety. Tyto hodnoty jsou pro vnitřní komunikaci jednotlivých modulů bloku CC-IIF dostačující.

[1]Stránky projektu Liberouter: https://www.liberouter.org/

Kapitola 3

Návrh řešení a implementace

Tato kapitola popisuje implementaci vlastního filtračního modulu pro zachycení odposlouchávaného provozu. S ohledem na celkovou architekturu LI systému se tedy jedná o blok CC-IIF. V kapitole budou popsány moduly CCCId pro přebírání zpráv z MF a jejich předání do filtrovacího modulu, vlastní filtrovací modul filterd a modul INI3d pro odesílání zachycené komunikace zpět do MF.

Řešení pro zpracování zákonných odposlechů je navrženo jako systém vzájemně komunikujících procesů, mezi něž jsou rozdělený úkoly jako komunikace se SLIS, vlastní filtrování provozu a hlášení zachycených paketů. Stěžejní část bloku CC-IIF, program filterd, počítá s nasazením na vysokorychlostních sítích dosahujících propustnosti v řádech desítek Gb/s. Zpracování tohoto množství paketů by v pouze v softwaru bylo velmi obtížné. Návrh tedy počítá s hardwarovou akcelerací pomocí síťové karty s firmware podporujícím SDM. Ten by měl postupně podle požadavků ze softwarové části omezit datový tok pouze na zájmový provoz. Implementace celého řešení je pak rozdělena do dvou vývojových fází.

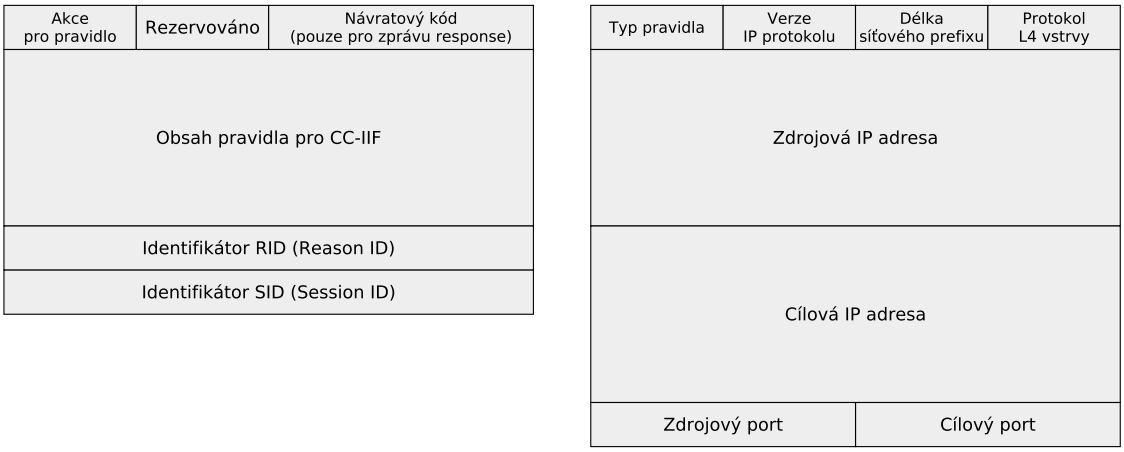
3.1 Komunikační protokoly bloku CC-IIF se SLIS

Jak je uvedeno v předchozí kapitole, blok CC-IIF je součástí většího systému komunikujících celků. Těm buď poskytuje zachycený obsah nebo upravuje nastavení svých filtrů na základě zpráv z AF. Rozhraní, jež slouží k této komunikaci jsou CCCI a INI3. V této sekci budou popsány formáty zpráv, které slouží ke komunikaci s tímto blokem pomocí uvedených rozhraní

3.1.1 Zprávy CCCI

Tyto zprávy slouží k předávání informací od CCTF o započetí nebo ukončení odposlechu. Zprávy jsou předávány binárním protokolem a obsahují identifikátory potřebné ke klasifikaci zájmových dat při zachytávání z provozu a spojení zachycených dat s odposlouchávaným cílem. Tyto zprávy pak slouži k nastavení filtrovacích tabulek.

Na obrázku 3.1 je znázorněn formát předávané zprávy. Ten obsahuje právě informace pro nastavení filtrovacích tabulek (pole *Akce pro pravidlo*) a identifikátory potřebné pro zpracování zachycených dat v MF (pole SID a RID). Pole *Pravidlo* pak obsahuje strukturu samotného pravidla pro zachytávání zájmového provozu. V tom jsou na základě typu pravidla vyplněna všechna potřebná pole (IP adresy, porty a protokol L4 vrstvy pro zachytávání



Obrázek 3.1: Zpráva protokolu CCCI: celá zpráva (vlevo) a obsah pravidla (vpravo)

konkrétního spojení, délka prefixu a IP adresa pro zachytávání veškeré komunikace v dané síti apod.). V současné podobě jsou tímto protokolem podporovány pouze dva typy pravidel. V prvním případě se jedná o požadavek na odposlech sítě s určitou maskou. Pravidlo tohoto typu je klasifikováno hodnotou 0 v poli *Typ pravidla*. Druhé pravidlo pak požaduje odposlech konkrétního síťového toku identifikovaného pěticí IP adresa zdroje, IP adresa cíle, L4 port zdroje, L4 port cíle a protokol L4 vrstvy. *Typ pravidla* pak obsahuje hodnotu 1. Pole návratový kód se objevuje pouze při odpovědi (dále označované jako zpráva typu response) odesílané zpět do CCTF a slouží k ověření, zda bylo pravidlo správně přijato. Ve zpravě s požadavkem (zpráva typu request) toto pole zůstává nevyplněné.

Obrázek 3.2: Zpráva protokolu INI3 verze 1

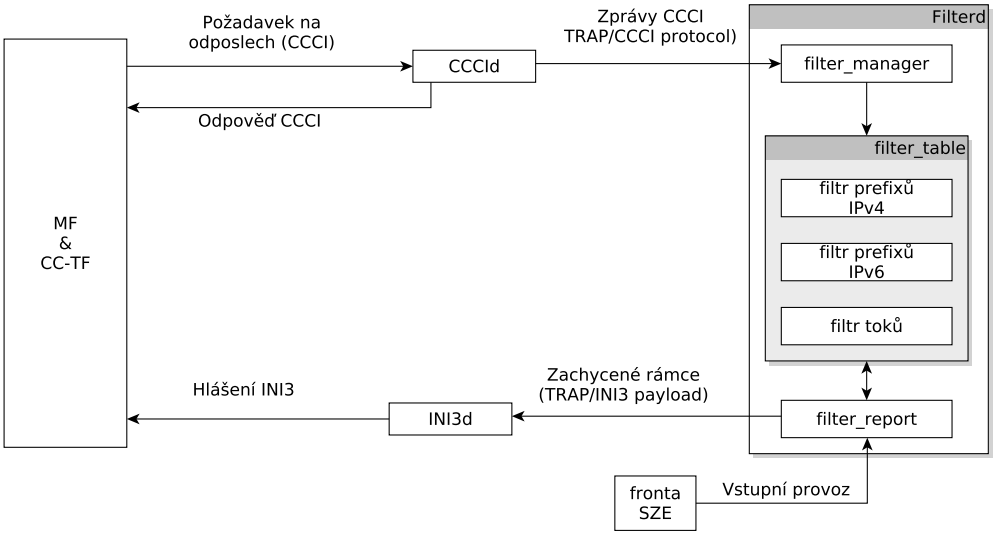
3.1.2 Zprávy INI3

Zprávy INI3 slouží k předávání zachycených dat zpět do MF. Společně se zachycenými daty jsou tímto rozhraním předávány i informace o rozhraní, na kterém byla data zachycena, časové razítko s dobou zachycení dat a identifikátory RID příslušící k danému rámci.

MF pak na základě těchto zpráv a informací z bloku IRI-IIF spojuje zachycená data s odposlouchávaným cílem. Formát těchto informací je znázorněn na obrázku 3.2.

3.2 1. fáze vývoje – Základní funkce CC-IIF

V první fázi jsou implementovány jednotlivé funkční bloky - CCCId s podporou rozhraní TRAP, filterd a INI3d. V této fázi je možné testovat základní funkčnost celé soustavy a dále je možné ověřit propustnost filtru pouze v softwarové podobě. Vizualizovaná podoba návrhu je znázorněna na obrázku 3.3. Ta ukazuje jednotlivé moduly bloku CC-IIF a jejich vzájemné propojení. Dále uvádí zprávy používané v komunikaci těchto bloků. V obrázku je podrobněji rozkreslen modul filterd, ve kterém je i naznačena vnitřní struktura rozdělení činnosti mezi vlákna, a také struktura tabulky pravidel.



Obrázek 3.3: Návrh 1. fáze implementace filtrovací sestavy

3.2.1 CCCId - démon pro zpracování požadavků na odposlechy

První částí celé sestavy je démon CCCId, který zpracovává požadavky na odposlech získané ze SLIS. Zpracování požadavku v tomto případě obnáší propagaci požadavku do následujícího prvku filterd. Ten pak požadavek zařadí do patřičné filtrovací tabulky. Zpráva je předána skrze rozhraní TRAP. CCCId do zprávy samotné nijak nezasahuje. Přenesená zpráva odpovídá formátu protokolu CCCI: Zpráva typu request. V této zprávě jsou uvedeny identifikátory SID (Session ID) a RID (Reason ID), akce, zda má být požadavek na odposlech přidán nebo odebrán, a vlastní obsah pravidla podle specifikace. CCCId po odeslání pak jen potvrdí přijetí pravidla.

3.2.2 Filterd - démon pro filtrování síťového provozu

Hlavní částí zpracování provozu je démon filterd, který se stará o zachycení rámců ze síťového provozu na základě pravidel získaných z démona CCCId. Zpracování rámců je nutné provádět v reálném čase. Program je proto rozdělen do dvou typů vláken - `filter_manager`

a `filter_report`. Vlákno `filter_manager` je v programu pouze jedno a stará se o příjem zpráv od CCCId. Na základě těchto zpráv pak provádí úpravy filtrovacích tabulek. Toto vlákno bude dále sloužit pro komunikaci se softwarovým řadičem SDM, kterému bude předávat pravidla o provozu, jenž má být zahozen již v hardwaru.

Vlákna `filter_report` přijímají rámce ze síťového provozu pomocí rozhraní SZE a následně zkouší tyto rámce přiřadit k některému z uložených pravidel. Pokud je pro rámec ve filtrovacích tabulkách nalezeno pravidlo, je následně vytvořena zpráva podle protokolu INI3Payload ve verzi 1. V ní jsou uvedeny příslušné RID, rozhraní, na kterém byl rámec zachycen, a časové razítko zachycení paketu s přesností v řádu nanosekund. Pak jsou k této zprávě připojena data v podobě L2 rámce s odstraněnou preambulí a kontrolním součtem. Takto vytvořená zpráva je následně odeslána opět přes rozhraní TRAP do dalšího démona INI3d. Pokud pravidlo není nalezeno, je zpracovávaný rámec zahozen a vlákno vystaví požadavek na zahazování rámců k tomuto síťovému toku již v hardwaru. Tento požadavek je pak zpracován vláknem `filter_manager`.

3.2.3 Implementace filtrovacích tabulek

Filtrovací tabulky jsou v démonovi filterd nejvytíženější částí programu. Přiřazení zachyceného paketu pravidlu musí být pokud možno co nejrychlejší, aby bylo možné zpracovávat provoz i na vysokorychlostních sítích (sítě s propustností 40 až 100 Gb/s a více). V současné fázi, kdy je ve SLIS implementována podpora pouze pro pravidla typu 0 (rámce s daty, jejichž IP adresa odpovídá příslušnému prefixu) a 1 (konkrétní datový tok identifikovaný pomocí pětice IP adresa zdroje, IP adresa cíle, L4 port zdroje, L4 port cíle a L4 protokol) jsou ve filtrovací sadě implementovány dva typy tabulek. Pro první typ pravidel je implementováno pole struktur, jehož položky obsahují odposlouchávaný prefix včetně jeho délky. K tomuto poli je přiřazena sada bitových polí s maskami pro všechny délky prefixů (0 - 32 pro IPv4 a 0 - 128 pro IPv6). Pro urychlení procesu přiřazení k prefixu je nad polem implementován algoritmus binárního vyhledávání. Pro tyto účely je pole seřazeno vzestupně podle IP adres. Pokud by se prefixy překrývaly, jsou pak ještě dodatečně řazeny podle jejich délky. Pole je pak vytvořeno v programu pro každou verzi IP protokolu zvlášť.

Síťové toky, jak je již zmíněno výše, jsou rozlišovány pěticí identifikátorů. Aby bylo možné tyto pravidla snadno spravovat, je pro ně využita hashovací tabulka. Ta zajišťuje, že lze na základě této sady identifikátorů jednoznačně, s ohledem na kolize, vyhledat příslušný tok. Aby se však zabránilo řetězení v případě kolizí a mohla tak být zaručena konstantní časová složitost v případě přístupu k toku, je využit mechanismus kukaččího vyhazování (cuckoo hashing). Tento mechanismus spočívá ve využití více hashovacích funkcí pro výpočet pozice dat. Při vkládání je nejprve ověřenlo, zda je pozice vypočítaná první hashovací funkcí volná. Pokud ano, pak je položka vložena na tuto pozici stejně jakov běžné hashovací tabulce. Je-li však tato pozice obsazena, jsou data, která tuto pozici obsazují, dočasně vyjmuta z tabulky a nahrazena nově vkládanou položkou. Pro data, která byla vyjmuta, je ověřena pozice vypočítaná další hashovací funkcí v pořadí a je aplikován stejný postup. Operace pokračuje do doby, než je možné bezpečně uložit všechny položky. Při tomto mechanismu je sice doba vkládání položky prodloužena o přesun stávajících dat do vhodných pozic, vyhledání je však na úrovni právě konstantní časové složitosti, kde doba odpovídá rychlosti výpočtu hashí toku podle příslušných algoritmu a porovnání klíče uloženého na vypočítané pozici.

V případě, že by nebylo nalezeno vhodné místo v tabulce a hrozila by ztráta uložených pravidel, je společně s touto tabulkou implementována stash v podobě jednosměrného seznamu. Ten sice sráží časovou složitost vyhledání položky v nejhorších případech na lineární, avšak k této situaci by mělo docházet velmi zřídka za předpokladu, že klíč k ukládaným pravidlům je v bytové podobě dostatečně unikátní a hashovací funkce použité v tabulce (RS, DJBv2 a BKDR) nejsou příliš kolizní. Algoritmy jednotlivých hashovacích funkcí jsou pak uvedeny v příloze B.

Kvůli přístupu ze dvou různých vláken by docházelo k problémům s ochranou paměti při zápisu nové položky nebo odebrání již neplatné položky. Mohlo by také docházet k dočasným ztrátám dat během operace přidávání nové položky, kdy jsou stávající data přesouvána. Použití zámků je však pro tento účel nežádoucí. Režie systému při jejich použití by totiž značně brzdila zpracování rámců. Díky tomu, že nad filtrovací sadou vždy operují nejvýše dvě vlákna, vlákno typu `filter_manager` a jedno vlákno typu `filter_report`, je možné použití zámků předejít. Metoda, která je k tomuto účelu použitá, spočívá v simulovaném běhu operace vložení. V tomto postupu vlákno `filter_manager` nejdříve provede operaci vložení nanečisto, kdy si zjistí, jaké položky by vyžadovaly přesun a zda je vůbec možné novou položku vložit. Jestliže je operace vložení možná, pak je zjištěný postup přesunů zpětně aplikován v opačném pořadí, než byl vytvořen. Tento postup zaručí, že v případě souběžného přístupu je v nejhorším případě vyhledávaná položka v daný okamžik v tabulce logicky zdvojená, tzn. je na ni odkazováno ze dvou pozic, ale vždy dostupná. Nová položka je do tabulky vložena až jako poslední. Jestliže by vložení nebylo možné, pak je výsledek simulované operace zahozen a uložena data jsou ponechána na svých pozicích. Nová položka je pak automaticky uložena do stashe.

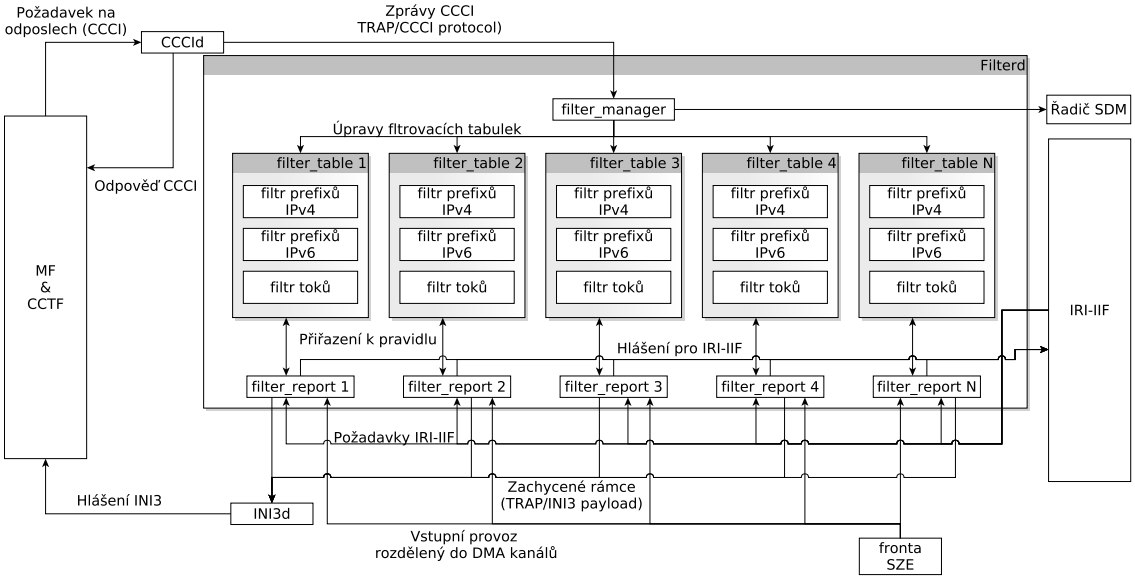
3.2.4 INI3d - démon pro hlášení zachycených paketů

Démon INI3d je posledním článkem celé sestavy a slouží k odeslání zachyceného rámce k dalšímu zpracování ve SLIS. Tento démon přijímá rámce zachycené ve filterd a přeposílá je zpět do bloku MF v podobě datového toku TCP. Pokud by došlo ke ztrátě spojení se SLIS nebo ke spojení nemohlo dojít při spuštění, jsou zachytávaná data ukládána do souboru. Tento soubor slouží k záloze do doby, než bude spojení obnoveno. Pak je tento soubor odeslán a činnost démona INI3d pokračuje běžným způsobem.

3.3 2. fáze vývoje – spojení s IRI-IIF a zprovoznění SDM

Druhá fáze implementace je navržena tak, aby co nejvíce využívala schopností hardwarové akcelerace. Jedním s těchto kroků je navýšení počtu vláken `filter_report` zpracovávajících vstupní provoz. Ta by měla být připojena k DMA kanálům síťové karty a odebírat provoz podle toho, jak jej bude hardware rozdělovat. Tuto funkci již podporuje například firmware HANIC[1] vyvíjený v rámci projektu Liberouter, který je schopen rozdělovat provoz na jednotlivé DMA kanály rovnoměrně systémem round-robin nebo podle výpočtu hashe. Díky tomu lze určit, jaká část provozu bude přicházet na jednotlivá vlákna a lze tedy pro ně stanovit přesnou konfiguraci. Pro vlákno `filter_manager` to však znamená nutnost přiřazovat pravidla pro odposlechy na základě této konfigurace. Je zde také riziko, že některý rámec bude zachycen nesprávným vláknem. V lepším případě bude rámec přiřazen a přeposlán dále, i když přijde z jiného než očekávaného zdroje, v horším případě bude rámec zahozen, přestože vlákno, které mělo rámec vyhodnotit pro něj má pravidlo ve své tabulce.

Zároveň bude filterd spolupracovat s podvozkem IRI-IIF, kam bude přeposílat zachycený provoz. IRI-IIF pak bude na základě vyhodnocení obdrženého provozu vydávat požadavky na úpravu filtrovacích tabulek podobně jako jsou požadavky od oprávněných orgánů. Obrázek 3.4 pak ukazuje jednotlivé moduly bloku CC-IIF s démonem filterd rozděleným na jednotlivá vlákna a propojení s blokem IRI-IIF.



Obrázek 3.4: Návrh 2. fáze implementace filtrovací sestavy

3.3.1 Předzpracování provozu pomocí SDM

Filtrovací modul byl od začátku navržen pro hardwarově akcelerované zpracování provozu, jelikož čistě softwarové zpracování, jak se později ukáže v sekci 4.2.1, není dostatečné pro provoz na rychlostech dosahujících až 100 Gb/s. Pro tento úkol bude použito právě SDM. Firmware bude sloužit k odchycení nezájmového provozu již v hardwaru a software tak bude moci zpracovávat pouze ty rámce, které má předávat dále, nebo které ještě nebyly klasifikovány. Toto chování by mělo poskytnout filtrování velké části provozu (podle [11] je možné zredukovat vstupní provoz až o 80% nezájmových dat), která by zbytečně zatěžovala software. Díky tomu by měl pak filterd dosáhnout vysoké propustnosti i v případě jednovláknového zpracování. Při zpracování vstupu pomocí více vláken je podle testů v sekci 4.2.1 filterd schopen zpracovat provoz odpovídající 20 až 30 Gb/s. S využitím SDM a započítáním redukce provozu 80% pak bude filterd schopen dosahovat bezztrátové propustnosti 100 Gb/s, na kterou je návrh cílen. Není však nutné filtrovat úplně všechny toky. Kdyby například software obdržel rámec příslušící k toku, který se na odposlouchávané lince objeví výjimečně, pak jeho filtrování není nutné. Softwarový řadič bude tedy uchovávat hlášení o označeném toku a pošle zprávu do hardwaru v okamžiku, kdy tato hlášení dosáhnou určité meze. Na vyřazení tohoto pravidla však bude řadič reagovat okamžitě, abychom zabránili ztrátě rámců náležících odposlouchávanému cíli.

[1]Hardware Accelerated Network Interface Card nebo HAshing Network Interface Card podle způsobu zpracování vstupního toku. Více informací lze najít například na https://www.liberouter.org/technologies/hanic/

3.3.2 Vazba s IRI-IIF

Myšlenkou tohoto propojení je využití démona filterd k tomu, aby blok IRI-IIF nemusel opakovat některou činnost pro stejná data. Příkladem může být výpočet hashí pro datový tok, který patří k výpočetně náročnější operaci a jehož opakování by mohlo mít další negativní vliv na výkonnost systému. Filterd tak bude mimo své hlavní činnosti sloužit ještě k nízkoúrovňovému předzpracování dat pro podvozek bloku IRI-IIF. Tomu bude na základě filtrovacích tabulek předávat vypočtené hashe pro tok, připravené offsety jednotlivých identifikačních polí rámce a v něm zapouzdřených protokolů a samotný rámec v podobě ukazatele do vstupní SZE fronty. IRI-IIF si přebere tato data ke zpracování a filterd pak obdrží zprávu, zda je o rámce tohoto toku zájem, či nikoliv. Podle odezvy pak filterd nastaví své filtrovací tabulky případně vydá pravidlo pro SDM. Vyhodnocení tohoto kroku musí být velmi rychlé, jelikož tato činnost souvisí se čtecími vlákny `filter_report`. Pokud by tato vlákna byla zdržována, mohlo by docházet ke ztrátě rámců. Dále je nutné zajistit úpravu filtrovacích tabulek podle požadavků IRI-IIF. Prozatímní řešení je ke každému vláknu přiřadit frontu pro ukládání těchto požadavků. Tato fronta bude vždy kontrolována vláknem `filter_manager`. Problémem tohoto řešení je stav, kdy by požadavky od IRI-IIF přicházely ve větším množství, než je lze zpracovat. Pak by nebylo možné provádět další úpravy. Řešením tohoto nedostatku může být zpracování pravidel po omezených dávkách např. 15 požadavků v jednom cyklu, pokud by nebylo velkým nedostatkem drobné zpoždění před zpracování další dávky.

Další komplikací s propojením filterd a IRI-IIF je využití SDM tak, aby svou funkcí vyhovovalo jak pro specifické požadavky od IRI-IIF, tak pro primárně zamýšlený účel. SDM totiž funguje tak, že propouští veškerý provoz a teprve na základě požadavků blokuje provoz nezájmový. To koresponduje s funkcí CC-IIF podle návrhu z první fáze, kdy si filterd upravuje vstupní provoz sám nebo na základě požadavků ze SLIS. IRI-IIF však potřebuje ke své činnosti pracovat s veškerým provozem. Filterd tak musí odesílat data, i když k nim nemá v tabulkách uvedené pravidlo. Zároveň ale dochází k tomu, že provoz, o který není zájem musí mít v tabulce zařazené pravidlo, aby nedocházelo k odesílání nežádoucího provozu. Po delší době běhu by tak mohlo dojít k zaplnění tabulky nebo ke kolizi identifikátorů RID.

Jednou možností pro řešení tohoto problému může být zavedení neaktivního timeoutu pro tento typ pravidel. Pravidla pro tento provoz by tak zůstávala v tabulce po dobu např. 30 sekund a pokud by toto pravidlo nebylo nijak pozměněno nebo použito, tak by vlákno `filter_manager` vydalo příkaz pro řadič SDM, aby byl odpovídající provoz blokován a poté by toto pravidlo bylo odebráno z tabulky. Druhým řešením je "obrácení"činnosti SDM. V tomto případě by SDM fungovalo tak, že by nepropouštělo žádný provoz a filterd by pak na základě pravidel pro přidání vydával pravidla pro povolení daného provozu. SDM by tak plnil funkci podobnou firewallu.

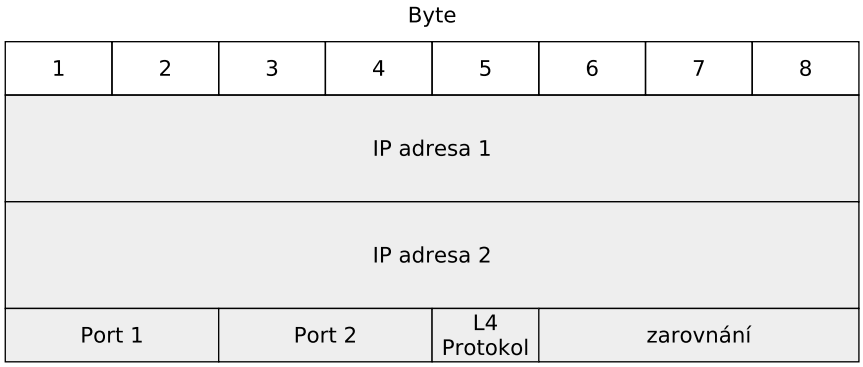
3.4 Základní implementace a její optimalizace

První verze navrhovaných modulů byla implementována podle návrhu v sekci 3.2. Implementace byla primárně cílena tak, aby moduly nevykonávaly žádnou činnost, pokud není nutná. Pro modul filterd to znamená, že má například zahazovat rámce, které neodpovídají standardním ethernetovým rámcům případně tagovaným rámcům standardu 802.1Q [10].

Dále měl filterd zahazovat rámce, které neobsahují data zapouzdřená protokolem IP a nebylo by tedy možné vytvořit správný vyhledávací klíč. Zpráva INI3 pro předání dat byla pak vytvářena pouze v okamžiku, kdy bylo možné data předat. Struktury implementované pro potřeby filtrů byly vytvořeny pro uchování jakýchkoli dat. Hashovací funkce tedy vypočítávaly hash z klíče po jednotlivých bytech, pokud funkce nevyžadovala jiné chování. Vyhledání a přiřazení IP adresy k danému prefixu bylo implementováno tak, aby algoritmus uměl zpracovat jak IPv4, tak IPv6 stejnou funkcí.

Takto implementovaný modul zabezpečuje, že rámec filtrem zpracován, pokud obsahuje data, ze kterých lze získat všechny potřebné identifikátory. Negativním dopadem tohoto řešení je však vysoké množství řídicích konstrukcí. Tyto konstrukce vyžadují nejen výpočet cíle skoku, ale i vyhodnocení predikce skoku. Obě tyto činnosti jsou pro procesor časově náročné a silně degradují výkonnost zpracování vstupního proudu dat. Výkonnost takto implementovaného modulu filterd je pak vyhodnocena v sekci 4.2.1. Pro co nejrychlejší zpracování rámce je tedy nutné tyto konstrukce redukovat na minimální množství i za cenu ztráty přenositelnosti některých algoritmů nebo vykonání některých méně náročných operací, i když nebude jejich výsledek dále využit. Mezi ně patří například zmiňované vytvoření zprávy INI3, přestože bude pravděpodobně zahozena. Její vytvoření obnáší pouze přiřazení správných hodnot do příslušných polí což je ve srovnání s vyhodnocením skoku nenáročná operace. Při tomto řešení není nutné vyhodnocovat zda má být zpráva vytvořena a následně zjišťovat, kterým komponentám má být odeslána. Po této úpravě je tedy zpráva vytvořena vždy a zůstává pouze rozhodnutí, jestli zprávu odeslat, či nikoli.

Další častou činností je výpočet hashe z vyhledávacího klíče. Ten probíhá s každým zpracovávaným rámcem třikrát, pokaždé jinou hashovací funkcí, jelikož není předem známé, na které z vypočítaných pozic je vyhledávané pravidlo v danou chvíli uloženo. Pro zajištění zpracování co nejvyššího počtu rámců by měl být výpočet co nejrychlejší a tak nenáročný na procesor, jak to půjde. Je tedy vhodné využít takové výpočty, které nemají matematicky náročné operace a eliminovat nutnost skoků na minimální možnou míru. Skoky lze z výpočtu vyřadit poměrně jednoduše, jelikož velikost klíče je předem známá a za běhu programu se nemění. Hashovací funkce tedy nemusí procházet celý klíč v cyklu, ale může tento krok provést jako posloupnost shodných příkazů. Opět tak odpadá nutnost predikce skoku.



Obrázek 3.5: Struktura vyhledávacího klíče pro filtrovací tabulky

Dále je možné číst klíč po větších blocích, aby se redukoval počet nutných kroků. Klíč pro identifikaci jednoho síťového toku zabírá po zarovnání v paměti 40 bytů (struktura klíče je zobrazena na obrázku 3.5). Výpočet lze tedy provádět po blocích o velikosti osm bytů. Výsledná hash je tak spočítána pouze v pěti krocích. Dále lze optimalizovat výpočet správné pozice v tabulce tím, že bude velikost tabulky omezena pouze na mocniny čísla 2. Pak není nutné provádět výpočet výsledné pozice operací modulo, ale lze provést pouze vymaskování výsledné hashe operací AND s číslem *velikost tabulky - 1*. Další možností, jak zrychlit operace prováděné nad tabulkou, je využití jiných hashovacích funkcí. Musí ale splňovat požadavek na nízký počet kolizí a dostatečnou rychlost.

Kapitola 4

Testování

Testování implementovaného řešení probíhalo hlavně na části filterd, která bude v provozu nejvíce zatěžovaným prvkem celé soustavy. Byla testována datová propustnost na pěti velikostech síťových rámců (64 bytů, 128 bytů, 256 bytů, 512 bytů a 1518 bytů) s prázdnou tabulkou pravidel pro zjištění propustnosti filtru jako takového. Následně byly filtrovací tabulky postupně zaplňovány pravidly v násobcích 1024 položek. Použitá pravidla byla náhodně generována. Rozložení pravidel mezi pravidla pro síťové prefixy a pravidla pro síťové toky bylo přibližně 45:55 ve prospěch síťových toků. Vstupní data byla generována přístrojem Spirent Testcenter^{TM} SPT-2000A na rozhraní s propustností 10 Gb/s. Využití filtrovacích tabulek pak bylo testováno mimo implementovaný program. Zde byly sledovány možnosti zaplnění hashovací tabulky se třemi hashovacími funkcemi samostatně a s využitím stashe o 10, 25, 50 položkách. Testovacím strojem pak byl server ANT-3[1].

4.1 Metodika testování

Pro otestování propustnosti byly zvoleny dvě metody - ztrátovost paketů při maximálním vytížení linky a nalezení maximální bezztrátové propustnosti [3]. Testování probíhalo se zapnutými optimalizacemi překladače gcc O3. Využití filtrovacích tabulek bylo testováno pomocí zaplnění tabulky náhodně generovanými daty, které svou strukturou odpovídají datům ukládaným v reálné verzi programu, do prvního okamžiku, kdy nebylo možné data vložit. V testech byla postupně testována samostatná tabulka a následně byla přidána stash o velikostech 10, 25 a 50 položek.

4.1.1 Ztrátovost paketů na maximálním vytížení linky

V tomto testu byl generován provoz na maximální kapacitě rozhraní po dobu tří minut. Po uplynutí doby generování paketů byly z hardwarových čítačů pomocí programu `ibufctl` odebrány statistiky o celkovém množství paketů a množství ztracených paketů. V každé konfiguraci bylo provedeno deset testů, ze kterých byla poté odebrána nejvyšší a nejmenší hodnota, jež by mohly zkreslovat výsledný průběh. Výsledné hodnoty byly pak použity pro výpočet průměrné ztrátovosti. Pro jedno vláknovou konfiguraci bylo zvoleno testování na platformě NetCOPE[12] s firmware NIC_CV2_10G2 verzi 1.0. Použitý firmware však neumožňuje vícevláknové zpracování vstupu. Pro ověření propustnosti ve vícevláknové

[1] procesor Intel Xeon E5620@2.40 GHz, 4 jádra, podpora zpracování 2 vláken zároveň (Intel Hyperthreading), 24 GB RAM, systém CentOS 5.3, jádro 2.6.18

konfiguraci tak bylo použito softwarové řešení, kde byl za pomocí knihovny libpcap[2] nahrán každému vláknu soubor s rámci do paměti. Tento soubor pak byl jednotlivými vlákny cyklicky čten po dobu jedné minuty. Vstupní soubor použitý pro tento test obsahuje většinou velmi krátké rámce. Testuje tedy jednu z nejčastějších složek síťového provozu.

4.1.2 Nejvyšší dosažitelná propustnost bez ztráty paketů

V tomto testu bylo cílem nalézt takové zatížení rozhraní, kdy nedochází k zahazování rámců. Pro dosažení výsledku byla zvolena metoda půlení intervalu a po nalezení přibližného výsledku bylo inkrementálně zvyšováno zatížení, dokud nebyla nalezena hodnota, kdy došlo k zahození. S každou velikostí rámce byly provedeny tři testy po 1 minutě. V každém testu bylo cílem přiblížit se co nejvíce hranici zahození rámce. Po nalezení nejvyšší možné propustnosti byla tato hodnota ověřena zatížením na dobu tří minut. Po provedení všech testů byly dosažené výsledky pro jednotlivé velikosti zprůměrovány a s každou průměrnou rychlostí byl proveden test po dobu dalších tří minut.

4.1.3 Zaplnění filtrovacích tabulek

Testování bylo provedeno nejdříve na samotné hashovací tabulce. V těchto testech bylo cílem zjistit, kolik položek je možné uložit do tabulky, než bude považována za zaplněnou. Tato situace nastane v okamžiku, kdy nelze vložit další položku, i kdyby došlo k přesunutí již stávajících položek do jejich náhradních pozic, případně byla již zaplněná i stash. Testy byly provedeny s každou jednotlivou hashovací funkcí, jejich dvojicemi a následně v plném počtu hashovacích funkcí. V druhém kroku byla hashovací tabulka v plné podobě testována s různou velikostí stashe. Zde bylo cílem najít takovou velikost stashe, kdy by tabulka dosáhla optimálního zaplnění pro požadovanou množinu dat. Pro samotné testy byla zvolena velikost hashovací tabulky 131 072 (2^{17}) položek jako výchozí. Velikost stashe byla pak v testech zvolena na 10, 25, 50 položek.

4.1.4 Rychlost výpočtu hashí

Testování rychlosti výpočtů hashí probíhalo s hashovacími funkcemi oddělenými od všech struktur. Funkce byly optimalizovány na konkrétní použitý klíč, jak bylo nejvíce možné. Cílem bylo najít takové hashovací funkce, které byly dostatečně rychlé pro zpracování vstupního toku. Každá funkce byla testována po dobu jedné minuty. Výsledek je poté prezentován průměrem ze sta po sobě jdoucích průběhů.

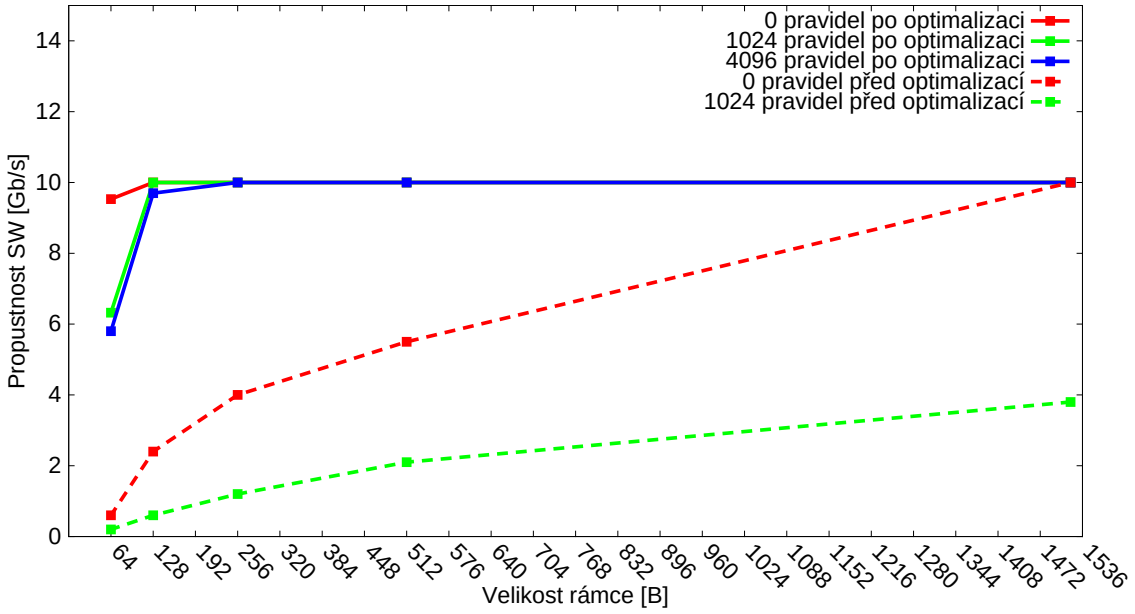
4.2 Získané výsledky

V této sekci jsou uvedeny výsledky z testů popsaných v předchozí části. Dále je zde přiložen i profil programu `filterd`, který ukazuje zastoupení jednotlivých operací v jeho běhu.

4.2.1 Propustnost modulu filterd a procento zahozených rámců

Na grafu 4.1 je zobrazena propustnost filtru při zpracování rámců uvedených délek. Tabulka 4.1 pak dále ukazuje ztrátovost rámců podle jejich délek. Vyznačené body ukazují délky, které byly použity v testech. Přerušovanou čárou jsou vyznačeny hodnoty z prvního

[2] www.tcpdump.org/



Obrázek 4.1: Propustnost modulu filterd

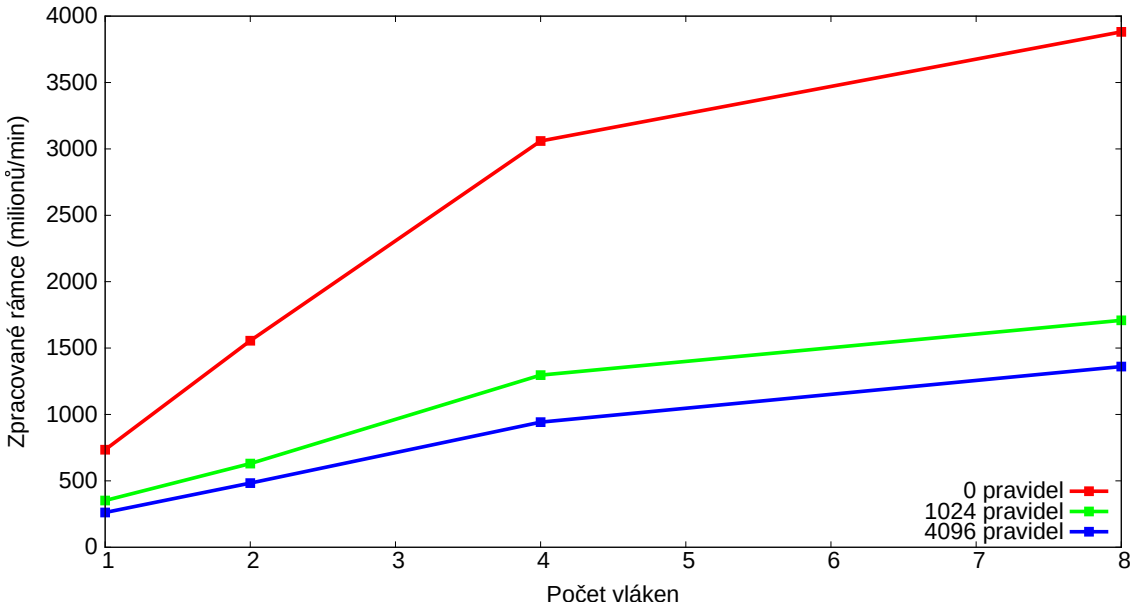
Délka rámce	**0 pravidel [rámce (%)]**	**1024 pravidel [rámce (%)]**	**4096 pravidel [rámce (%)]**
64	134 482 333 (5.02)	1 004 487 165 (37.5)	1 158 238 528 (43.24)
128	0 (0.0)	0 (0.0)	49 807 293 (3.28)
256	0 (0.0)	0 (0.0)	0 (0.0)
512	0 (0.0)	0 (0.0)	0 (0.0)
1518	0 (0.0)	0 (0.0)	0 (0.0)

Tabulka 4.1: Průměrná ztrátovost podle délky rámce při rychlosti linky 10 Gb/s

kola testů, tedy verze modulu filterd bez aplikovaných optimalizací specifických pro zpracování provozu. Na nich je možné vidět, že pokud uvažujeme výpočet hashovacích funkcí z každého jednotlivého bytu klasifikačního klíče a pokus o přiřazení do jednoho z prefixů, pak navržené řešení stěží postačuje na zpracování rámců délky 1518 bytů na rychlosti 10 Gb/s. Na nejkratších rámcích dosahuje neoptimalizovaný filterd propustnosti pouze 600 Mb/s při prázdných filtrovacích tabulkách.

Po aplikaci optimalizací je tento problém téměř eliminován. Mezi provedené optimalizace patří například změna způsobu výpočtu hashovací funkce, kdy není výpočet proveden po jednotlivých bytech, ale po větším bloku dat. Dále byly odstraněny některé řídicí konstrukce, které sice zbavovaly program zbytečných činností, avšak vlastní rozhodování je pro zpracování mnohem náročnější (podrobnější popis oprimalizací lze najít v sekci 3.4. I přesto není řešení schopné zpracovat provoz na rachlosti 10 Gb/s ani na prázdných filtrovacích tabulkách při nejkratších rámcích. S přidáním pravidel do tabulky začíná ztrátovost logaritmicky narůstat. To je hlavně způsobeno filtrem pracujícím se síťovými prefixy. Ten využívá právě algoritmu s logaritmickou složitostí (binární vyhledávání). Při větších délkách rámců se tento jev projevuje minimálně i s postupným zaplňováním tabulky.

Z předchozích grafů vyplývá, že je možné dosáhnout vysoké propustnosti pro přiřazení rámců jen softwarovým řešením, avšak toto řešení nebude dostačovat v případě vyšších nároků na zpracování. Hardwarová akcelerace je tedy nutností pro zajištění stabilního zpracování zájmových dat bez ztráty jediného rámce i na rychlostech vyšších než je zde testovaných 10 Gb/s. Propustnost s funkční hardwarovou akcelerací však nebylo možné otestovat, jelikož funkční verze konceptu SDM je vyvíjena pouze pro karty s propustností 100 Gb/s a tento hardware nebyl v době testování ještě dostupný. Dalším prvkem, který nebylo možné otestovat stejnou metodou, je zpracování vstupu pomocí více čtecích vláken. Tato vlastnost není správně podporována firmwarem na testované platformě. Pro základní testování propustnosti je sice možné použít firmware HANIC. Ten však nebylo možné správně uvést do provozu na testovacím stroji v aktuální verzi 2.0 a s verzí 1.1, která je na testovacím stroji dostupná, nebylo možné získat data o zahozených rámcích z hardwarových čítačů. Pro zisk dat nakonec byla využita softwarová metoda popsaná v sekci 4.1.1. Výsledek tohoto testu je zobrazen v grafu 4.2.



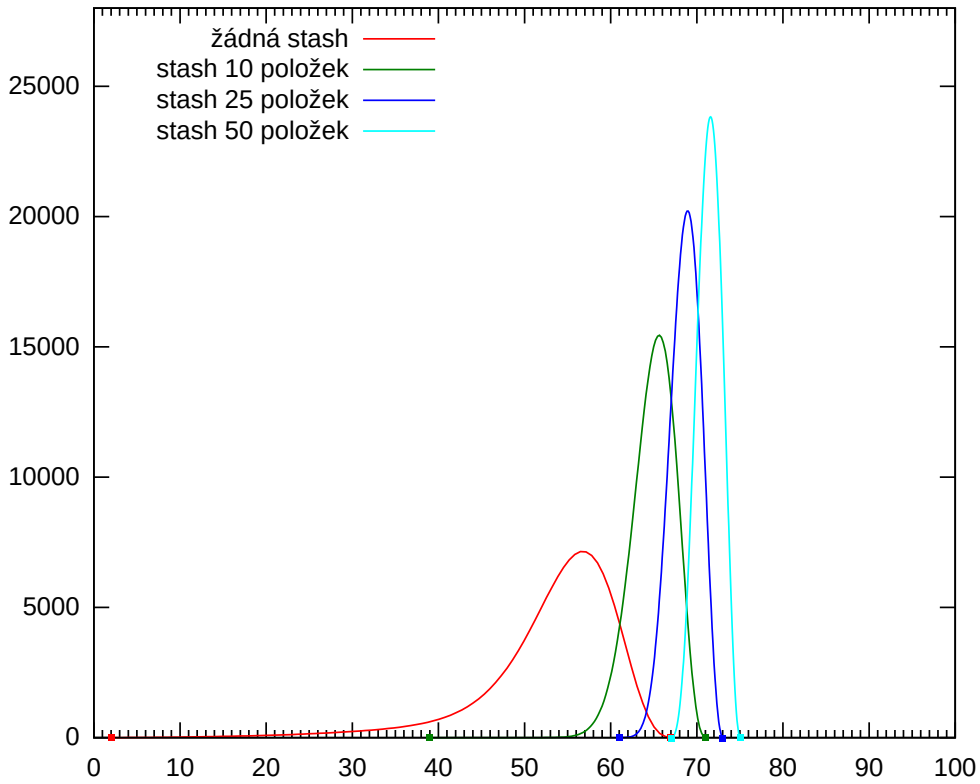
Obrázek 4.2: Výkonnost vícevláknového zpracování (krátké rámce, soubor pcap)

V grafu je tedy vidět, jak je řešení škálovatelné při zvýšení počtu vláken pro zpracování vstupu. Díky tomu, že program filterd za běhu využívá nejméně pět vláken (1 hlavní vlákno, 2 vlákna rozhraní TRAP - vstup a výstup, 1 vlákno `filter_manager` a 1 vlákno `filter_report`), je vidět téměř lineární nárůst výkonu do použití čtyř vláken `filter_report`. Rozdíl mezi použitím čtyř a osmi vláken již není tak velký, jelikož se zde začíná projevovat režie operačního systému, která brzdí zpracování vstupu. I tak je ale vidět, že vícevláknové zpracování vstupu je schopno pracovat i na vyšších rychlostech, než je testovaných 10 Gb/s.

4.2.2 Kapacita a zaplnění filtrovacích tabulek

Pro bezchybnou funkci filtrovacího modulu je potřebné zajistit, aby byla příslušná pravidla uchována v takové struktuře, která zajistí, aby nedošlo ke ztrátě pravidel za běhu. Samotná hashovací tabulka s kukaččím vyhazováním však tuto vlastnost nemá, proto je s ní

implementována dodatečná struktura (stash) uchovávající pravidla, která by normálně musela být odstraněna. Tato stash může mít negativní dopad na rychlost zpracování rámců. Proto musí být zvolena taková velikost obou struktur, aby stash byla používána pokud možno co nejméně a zároveň bylo možné uložit do tabulky co největší část z ukládané množiny dat.



Obrázek 4.3: Zaplnění filtrovacích tabulek s různou velikostí stashe

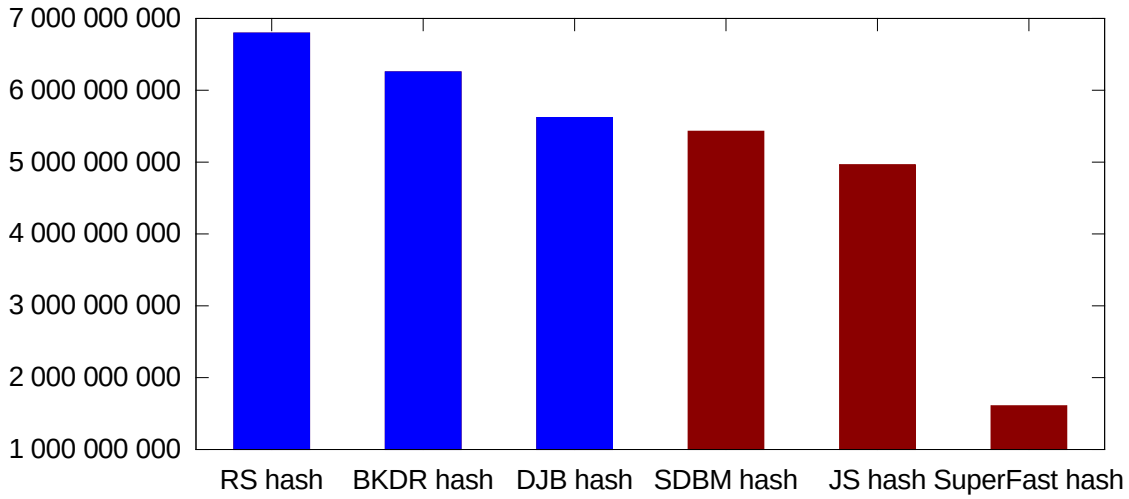
Graf 4.3 ukazuje maximální možnosti zaplnění hashovací tabulky do doby než dojde k takové kolizi klíčů, že nelze vložit další položku (tabulka se jeví jako zaplněná). Osa X grafu ukazuje procentuální zaplnění tabulky dosažených v jednotlivých běžích testu a osa Y pak počet běhů, které tohoto zaplnění dosáhly. Jednotlivé křivky ukazují tuto vlastnost jak pro tabulku samotnou, tak pro tabulku s velikostmi stashe 10, 25 a 50 položek. Pro každou křivku jsou pak vyznačené maximální a minimální dosažená zaplnění tabulky.

V optimálním případě by měl být rozsah mezi maximálním a minimálním zaplněním co nejmenší. Tento rozsah se dá regulovat právě velikostí stashe. S velikostí 50 položek se tabulka dostane dokonce na minimum 67% zaplnění, což je při množině 2^{16} identifikátorů RID dostačující ve všech případech. Takto velká stash by ale už mohla vést ke značné degradaci výkonu, jelikož je implementována pouze lineární strukturou. Zároveň tato stash má sloužit pouze jako rezerva, nikoli jako další zdroj dat. Ve výsledné implementaci je tedy nakonec zvolena velikost 25 položek. Ta poskytuje stabilně dostatečně velké zaplnění hashovací tabulky s tím, že stash bude využívána pouze ve výjimečných případech.

4.2.3 Rychlost výpočtu hashe identifikátoru toku

Vyhledání pravidla v hashovací tabulce je dalším kritickým místem činnosti filtrovacího modulu. Výpočet hashí pro přístup do tabulky musí probíhat pro každý přijatý rámec, a to třikrát. Zvolené hashovací funkce tak musí být dostatečně rychlé, aby jejich výpočet postačoval na požadovanou propustnost. Pro časové okno 3 minuty je to cca 2,7 miliardy rámců o velikosti 64 bytů a 146 milionů rámců pro 1518 bytů. Pokud jsou zvolené hashovací funkce schopné vypočítat toto množství trojic, pak by měla teoretická propustnost dosahovat 10 Gb/s.

V grafu 4.4 jsou zobrazeny testované hashovací funkce a počet hashí, které jsou schopny vypočítat samostatně za jednu minutu. Modrou barvou jsou označeny hashovací funkce zvolené ve výsledné implementaci. Funkce samotné jsou optimalizovány na zpracování konkrétního vstupního klíče. Díky tomu jsou schopny vypočítat toto množství. Zajímavý je výsledek funkce SuperFast hash, jenž je oproti ostatním funkcím řádově nižší. To je způsobeno tím, že funkce byla při testu ponechána ve své původní podobě. Ta zpracovává vstupní klíč po dvou bytech. Ostatní funkce jsou optimalizovány na zpracování po osmi bytech. Při pokusu o přizpůsobení této hashovací funkce na stejnou velikost zpracovávaného bloku se však výsledek výrazně zhoršil. Vliv na výsledek může mít v této funkci i další výpočetní krok pro vynucení *avalanche efektu*[3]. Tento výpočet sice umožní dosažení lepšího rozložení pro tuto hashovací funkci, pro účely vyhledávání a ukládání dat je však tento krok zbytečně náročným. Ostatní uvedené hashovací funkce tento výpočet nevyužívají. Algoritmus funkce SuperFastHash je pah uveden v příloze C.



Obrázek 4.4: Rychlost výpočtu jednotlivých hashovacích funkcí

4.2.4 Výsledny profil modulu filterd

V této sekci je uveden profil modulu filterd podle operací prováděných při zpracování vstupního provozu. Tento modul je totiž nejvytíženějším ze všech tří implementovaných částí

[3] Avalanche efekt je vlastnost hashovací funkce, která zajišťuje, že i při velmi malé změně vstupu např. jedno písmeno ve vstupním řetězci nebo i jeden bit dojde výrazné změně ve výstupu. Tato vlastnost je hojně využívána především v hashovacích funkcích používaných v kryptografii (algoritmy MD5, SHA a další).

a vyžaduje tedy nejvíce pozornosti při optimalizacích. Profil detekuje místa, kde modul tráví nejdelší dobu, a tím podá informací o operacích, které by měly být v dalších fázích vývoje optimalizovány nebo upraveny. Profil byl získán programem `valgrind` a jeho nástrojem `callgrind`[4]. Všechny hodnoty ukazují procentuální zastoupení těchto operací v běhu programu. V tabulkách 4.2 a 4.3 jsou uvedeny profily `filter_report` zpracovávající vstupní provoz a jeho funkce pro vyhledání a přiřazení k pravidlu`filter_match`. Tabulky ukazují profil pro prázdné filtrovací struktury a následně pro struktury, které jsou částečně zaplněné (v tomto případě byla zvolena hodnota 1024 pravidel).

	0 pravidel	1024 pravidel
Připojení k rozhraní SZE	16%	8%
Čtení ze vstupního rozhraní SZE	17%	8%
Vytvoření vyhledávacího klíče	16%	13%
Vyhledání a přiřazení k pravidlu	31 %	60%

Tabulka 4.2: Profil běhu vlákna `filter_report`

	0 pravidel	1024 pravidel
Výpočet hashí z vyhledávacího klíče	48%	12%
Vyhledání položky v hashovací tabulce	13%	3%
Vyhledání položky na základě prefixu	5%	66%

Tabulka 4.3: Profil vyhledávací funkce `filter_match`

Podle tabulek je tedy zřejmé, že nejnáročnější částí programu je samotné přiřazení položky k pravidlu. Největší podíl na této činnosti má přiřazení k pravidlu na základě prefixu kvůli algoritmu binárního vyhledávání. Ten je nutné provést pro každou IP adresu. Při maximálním počtu pravidel tj. 2^{16} by musel filterd projít 32 položek (16 pro IP adresu zdroje a 16 pro IP adresu cíle) v nejhorším případě.

[4] http://valgrind.org/docs/manual/cl-manual.html

Kapitola 5

Závěr

V práci byl navržen systém pro zpracování síťového provozu za účelem zákonných odposlechů. Návrh popisoval jak provedení vlastního modulu, tak jeho integraci do celého systému pro odposlechy a jeho vazbu na další komponenty v budoucím vývoji. Navrhovaný modul byl pak implementován v jeho první verzi, která byla poté testována na schopnost zpracovávat síťový provoz při rychlosti linky 10 Gb/s. Z výše uvedených testů je zřejmé, že implementované řešení i při optimalizacích má stále nedostatečný výkon především na menších velikostech rámců. Vzhledem k současnému typu provozu, kde jsou v největší míře zastoupeny buď právě velmi krátké rámce (hlavičky různých protokolů, rámce kontrolující životnost serverů apod.), nebo naopak dlouhé rámce, by mohlo dojít ke ztrátám rámců důležitých pro činnost celého LI systému. V implementaci další fáze je tedy nutné se zaměřit na tyto nedostatky a pokusit se je odstranit nebo alespoň co nejvíce zmírnit.

Jedním z kritických míst je právě klasifikace vstupního provozu na základě prefixů, jak ukazují tabulky uvedené v sekci 4.2.4. V druhé fázi vývoje by tedy bylo výhodné vytvořit takový algoritmus, který by dokázal adresu přiřadit k prefixu přesné délky a provedl tuto operaci v co nejkratším čase a ideální s konstantní časovou složitostí. Jednou z možností je např. tabulka, která by uchovávala data pro nejčastější délku prefixu v dané síti a na zbylé varianty by mohl být použit stejný algoritmus jako teď. Výpočet hashe pro tuhle tabulku by pak klidně mohl zabezpečit hardware například tím, že by společně s rámcem předával již vypočtenou hash.

Další optimalizací již zmíněnou v druhé fázi vývoje je vícevláknové zpracování vstupu. S ním v kombinaci firmware na použité kartě by bylo možné rozložit zpracování vstupních dat na základě například hardwarově počítané hashe. To by zajistilo, že každé vlákno pak obdrží pouze část provozu a jedno vlákno tak nebude muset zpracovávat všechna data přicházející na vstup. Navíc s využitím SDM bude možné odfiltrovat podstatnou část nezájmového provozu a software tak bude schopen zpracovávat zájmová data bez zbytečné zátěže.

Literatura

[1] Baker, F.; Foster, B.; Sharp, C.: Cisco Architecture for Lawful Intercept in IP Networks. RFC 3924 (Informational), Říjen 2004.
URL http://www.ietf.org/rfc/rfc3924.txt

[2] Bartoš, V.; Žádník, M.; Čejka, T.: Nemea: Framework for stream-wise analysis of network traffic. Technická zpráva, 2013.
URL http://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf

[3] Bradner, S.; McQuaid, J.: Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational), Březen 1999, updated by RFCs 6201, 6815.
URL http://www.ietf.org/rfc/rfc2544.txt

[4] Deering, S.; Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Prosinec 1998, updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.
URL http://www.ietf.org/rfc/rfc2460.txt

[5] European Telecommunications Standards Institute: *ETSI TR 101 944: Telecommunications security; Lawful Interception (LI); Issues on IP Interception.* 12 2001, v1.1.2.

[6] European Telecommunications Standards Institute: *ETSI TR 101 943: Telecommunications security; Lawful Interception (LI); Concepts of Interception in a Generic Network Architecture.* 11 2006, v2.2.1.

[7] European Telecommunications Standards Institute: *ETSI TR 102 528: Lawful Interception; Interception domain Architecture for IP networks.* 10 2006, v1.1.1.

[8] European Telecommunications Standards Institute: *ETSI TR 101 671: Lawful Interception (LI); Handover interface for the lawful interception of telecommunications traffic.* 11 2011, v3.9.1.

[9] European Telecommunications Standards Institute: *ETSI TR 101 331: Telecommunications security; Lawful Interception; Requirements of Law Enforcement Agencies.* 02 2014, v1.4.1.

[10] Institute of Electrical and Electronics Engineers: *IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks.* 05 2011.

[11] Kekely, L.: *Hardwarová akcelerace aplikací pro monitorování a bezpečnost vysokorychlostních sítí.* Diplomová práce, FIT VUT v Brně, Brno, 2013.

[12] Martínek, T.; Košek, M.: NetCOPE: Platform for Rapid Development of Network Applications. In *2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, s. 219–224, doi:http://dx.doi.org/10.1109/ddecs.2008.4538789.

[13] Matoušek, P.: *Síťové aplikace a správa sítí: Architektura sítí, adresování, konfigurace TCP/IP*. FIT VUT v Brně, Brno, 2011.

[14] Polčák, L.; Kramoliš, P.; Kajan, M.; aj.: Architektura systému pro zákonné odposlechy. Technická zpráva, 2011.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9829

[15] Postel, J.: Internet Protocol. RFC 791 (INTERNET STANDARD), Září 1981, updated by RFCs 1349, 2474, 6864.
URL http://www.ietf.org/rfc/rfc791.txt

Příloha A

Obsah CD

`BP_text` Zdrojové kódy a soubory pro sazbu textu BP

`CCCI_sim` Program pro simulaci činnosti CCCId

`filterd` Implementace programu pro filtrování síťového provozu pro zákonné odposlechy

`INI3d` Implementace programu pro odesílání odposlechnutého provozu do systému pro zákonné odposlechy

Příloha B

Hashovací funkce použité ve filtrovacích tabulkách

B.1: Robert Sedgewick's hash

```
unsigned int RSHash(char* str, unsigned int len)
{
    unsigned int b = 378551;
    unsigned int a = 63689;
    unsigned int hash = 0;

    for( unsigned int i = 0; i < len; str++, i++) {
        hash = hash * a + (*str);
        a    = a * b;
    }

    return hash;
}
```

B.2: DJB hash verze 2 (Daniel J. Bernstein)

```
unsigned int DJBHash(char* str, unsigned int len)
{
    unsigned int hash = 5381;
    unsigned int i    = 0;

    for (unsigned int i = 0; i < len; str++, i++) {
        hash = ((hash << 5) + hash) + (*str);
    }

    return hash;
}
```

B.3: BKDR hash (Brian Kernighan a Dennis Ritchie)

```
unsigned int BKDRHash(char* str, unsigned int len)
{
    unsigned int seed = 131; /* 31 131 1313 13131 ...*/
    unsigned int hash = 0;

    for (unsigned int i = 0; i < len; str++, i++) {
        hash = (hash * seed) + (*str);
    }

    return hash;
}
```

Příloha C

Algoritmus SuperFastHash

```
unsigned long SuperFast_hash(char* str, unsigned int len)
{
    uint64_t hash = len, tmp;
    int rem;

    if (len <= 0 || str == NULL) return 0;

    rem = len & 3;
    len >>= 2;

    /* Main loop */
    for (;len > 0; len--) {
        hash  += get16bits (str);
        tmp    = (get16bits (str+2) << 11) ^ hash;
        hash   = (hash << 16) ^ tmp;
        str  += 2*sizeof (uint16_t);
        hash  += hash >> 11;
    }

    /* Handle end cases */
    switch (rem) {
        case 3: hash += get16bits (str);
                hash ^= hash << 16;
                hash ^= str[sizeof (uint16_t)] << 18;
                hash += hash >> 11;
                break;
        case 2: hash += get16bits (str);
                hash ^= hash << 11;
                hash += hash >> 17;
                break;
        case 1: hash += *str;
                hash ^= hash << 10;
                hash += hash >> 1;
    }
```

```
    /* Force "avalanching" of final 127 bits */
    hash ^= hash << 3;
    hash += hash >> 5;
    hash ^= hash << 4;
    hash += hash >> 17;
    hash ^= hash << 25;
    hash += hash >> 6;

    return hash;
}
```